



Buenas prácticas de programación en Python

LECCIÓN 5: CONTROL DE VERSIONES GIT Y GITHUB

ÍNDICE

CONTROL DE VERSIONES GIT Y GITHUB

Presentación y objetivos.....	2
2. Control de versiones git.....	3
1.1 Terminología	4
1.2 Arquitecturas de almacenamiento	6
3. Control de versiones GitHub y su integración en visual studio code	8
2.1 Sincronización proyecto GitHub en Visual Studio Code.....	10
2.2 Creación de ficheros y subida a GitHub (commit y push).....	14
2.3 Descarga y actualización del proyecto (Pull)	19
4. Conclusiones generales	20
5. Puntos clave	20

Control de versiones Git y GitHub

PRESENTACIÓN Y OBJETIVOS

En esta lección aprenderemos en qué consiste los Sistemas de Control de Versiones y por qué su utilidad lo ha convertido en una herramienta fundamental e imprescindible para todo desarrollador de software.

Adicionalmente, estudiaremos el Sistema de Control de Versiones Git y GitHub y cómo integrar este último en el entorno de desarrollo Visual Studio Code, así como las herramientas principales para poder crear, clonar, actualizar y descargar contenido de un repositorio en GitHub.



Objetivos

- Conocer qué son los Sistemas de Control de Versiones.
- Conocer cuáles son las características principales de los Sistemas de Control de Versiones.
- Saber crear un repositorio en GitHub.
- Clonar y gestionar repositorios de GitHub desde el IDE Visual Studio Code.

1. CONTROL DE VERSIONES GIT

Denominamos control de versiones a la gestión de diferentes modificaciones que se han realizado sobre los elementos de algún producto, ya sean códigos de programación o simples documentos de texto. Una versión hace referencia al estado en el que se encuentra dicho elemento en un momento dado de su desarrollo o modificación.

Como podéis imaginar, un control de versiones puede ser realizado a mano. Una tarea tan simple como la de guardar registros en diferentes directorios o ficheros sobre el trabajo que estemos realizando. Puede parecer una tarea trivial pero, ¿os imagináis tener que guardar una copia para cada cambio que hagáis en un determinado documento?. Al principio, cabe la posibilidad de que sea posible llevar este registro de versiones a mano, ¿pero qué ocurriría cuando llevéis trabajando sobre un mismo proyecto durante un año? o peor aún, ¿y si más de una persona está trabajando sobre dicho proyecto?.

Ante este tipo de casuísticas, se aconseja disponer de herramientas que faciliten la tarea de gestionar las diferentes versiones de un proyecto. A estas herramientas se les conoce como sistema de control de versiones o VCS (de sus siglas en inglés **V**ersion **C**ontrol **S**ystem).

El control de versiones se utiliza principalmente en mundo de la informática, más específicamente, controlando las diferentes versiones de código fuente que se va desarrollando a lo largo del proyecto. De este modo, podemos contar con un sistema que nos registra cada cambio, inclusión o borrado de cualquier línea de código de nuestro proyecto, dando lugar a los sistemas de control de código fuente.

No obstante, las herramientas de control de versiones también son aplicables a otras áreas de interés como imágenes, documentos, etc.

Todo sistema de control de versiones que se precie debe proporcionar las siguientes características:

- Mecanismo de almacenamiento para los elementos que debe gestionar.
- Posibilidad para realizar cambios sobre los elementos proporcionados.
- Registro histórico de las acciones que se han ido realizando sobre todos y cada uno de los elementos almacenados.
- Generador de informes con los cambios que se han ido introduciendo entre las diferentes versiones.

1.1 Terminología

Actualmente podemos encontrar una gran variedad de sistemas de control de versiones. Antes de explicar uno de los más conocidos, es necesario mostrar y definir cuál es la terminología más utilizada en softwares de control de versiones:

- **Repositorio.** Es el lugar donde se almacenan los datos de nuestro proyecto, tanto los actualizados como sus diferentes versiones. A menudo se almacenan en un servidor.
- **Revisión.** Una revisión es una versión determinada de la información que se gestiona. A la última versión del proyecto se le suele identificar con el nombre **head**.
- **Etiquetas** o tags. Las etiquetas permiten reconocer fácilmente entre las revisiones más importantes que se han realizado a lo largo del proyecto.

- **Rama** o branch. Durante el desarrollo de un proyecto, uno de los módulos puede ser dividido de forma que existan dos copias iguales del estado del proyecto y que posteriormente, cada una de ellas avance de forma independiente siguiendo su propia línea de desarrollo. Cuando esto ocurre decimos que el proyecto tiene dos o más ramas. La principal ventaja, es que llegado el momento se pueden unificar las modificaciones que se han hecho en cada una de las ramas (merge), dando la posibilidad de comprobar que el contenido de las diferentes ramas funciona correctamente. De ser así, se unifican las diferentes ramificaciones de nuestro proyecto con la rama principal.
- **Fusión** (merge). Una fusión se encarga de unificar dos conjuntos de cambios que han sido realizados sobre uno o varios ficheros en una revisión unificada de dichos ficheros.
- **Publicar** (commit). Un commit ocurre cuando se han realizado uno o varios cambios sobre un fichero del proyecto. Un commit nos permite comentar cuáles han sido los cambios que se han realizado para, una vez subido al repositorio, tengamos un registro (a modo de diario) de las modificaciones que han ido sufriendo cada uno de los ficheros de nuestro proyecto.
- **Conflicto**. Un conflicto ocurre cuando el sistema de control de versiones no puede gestionar adecuadamente los cambios realizados por diferentes usuarios en un mismo archivo. Por ejemplo, si dos usuarios realizan modificaciones sobre una misma línea de un mismo fichero, surgirá un conflicto. En estos casos, el sistema no puede fusionar los cambios y serán los propios usuarios quienes deban tomar las decisiones oportunas para arreglar este conflicto. Una de las decisiones más comunes consiste en elegir los cambios realizados por uno de los usuarios y descartar el otro.
- **Actualización**. Integra los cambios que han sido realizados en el repositorio (por ejemplo, por otros usuarios) en la copia del proyecto que tenemos almacenada en local.

1.2 Arquitecturas de almacenamiento

Atendiendo a la estrategia seguida para almacenar las diferentes versiones de un proyecto, podemos identificar dos tipos de sistemas de control de versiones:

- **Distribuidos.** Cada usuario tiene su propio repositorio. Los diferentes repositorios pueden ser intercambiados y mezclados entre ellos. Algunos ejemplos de sistemas de control de versiones distribuidos son **Git** y **Mercurial**.

A continuación se detallan las principales ventajas de un sistema de control de versiones distribuido:

- Implican una mayor autonomía, ya que se necesita estar conectado a la red menos veces para realizar operaciones.
 - Si el repositorio remoto sufre alguna avería, los desarrolladores pueden seguir trabajando en local.
 - Dado que cada usuario perteneciente a un determinado proyecto tendrá una réplica local de la información del repositorio remoto, la información estará muy replicada. Esto implica que hay menos necesidad de hacer copias de seguridad y que el sistema tendrá menos problemas para recuperarse ante una posible incidencia.
 - El servidor en remoto necesita menos recursos que los que necesitaría un servidor centralizado, puesto que la mayor parte del trabajo se realiza en repositorios locales.
-
- **Centralizados.** Existe un repositorio centralizado que contiene todo el código, y uno o varios usuarios serán los únicos responsables del repositorio. Sacrificando la flexibilidad del sistema de control de versiones, se facilita por contra las tareas administrativas. Algunos ejemplos son **CVS** o **Subversion**.

A continuación se detallan las principales ventajas de un sistema de control de versiones centralizado:

- En los sistemas de control de versiones distribuidos hay un menor control a la hora de trabajar en equipo ya que no se tiene una versión centralizada (unificada) de lo que se ha ido realizando sobre el proyecto. Esto no ocurre en un sistema de control de versiones centralizado.
- Las diferentes versiones almacenadas son identificadas por medio de un número que indica su versión. Esto facilita enormemente la gestión del proyecto. Por contra, en los sistemas distribuidos no tiene sentido utilizar dicho número de identificación ya que cada repositorio (en local) tendrá sus propios números, dependiendo de los cambios que cada usuario haya realizado.

En esta lección aprenderemos sobre el uso de GitHub, una plataforma de desarrollo colaborativo para hospedar proyectos utilizando el sistema de control de versiones Git.



Presta atención

El código de los proyectos alojados en GitHub es almacenado usualmente de forma pública, aunque se permite la creación de proyectos privados. Hoy en día, GitHub continúa siendo la plataforma de referencia para la colaboración de proyectos Open Source.

2. CONTROL DE VERSIONES GITHUB Y SU INTEGRACIÓN EN VISUAL STUDIO CODE

En este apartado aprenderemos a integrar el sistema de control de versiones GitHub con nuestro IDE Visual Studio Code. Esta integración hace que dispongamos de una forma muy sencilla y amigable de un conjunto de herramientas que nos permitirá desarrollar nuestras implementaciones en Python, así como llevar un control de versiones actualizado sobre nuestro proyecto. Además, también se presenta la posibilidad de realizar proyectos de forma colaborativa.

En primer lugar, debemos crearnos una cuenta en la plataforma GitHub (<https://github.com/>)

Join GitHub

Create your account

Username *

Email address *

Password *

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more](#).

Email preferences

☐ Send me occasional product updates, announcements, and offers.

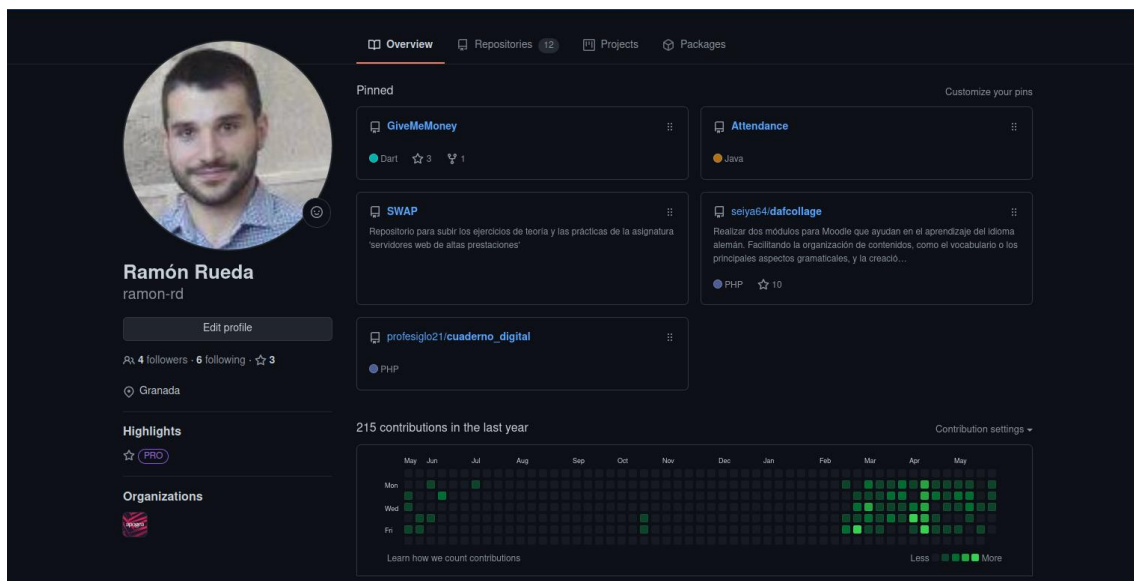
Verify your account

Solucione este rompecabezas para que sepamos que es una persona real

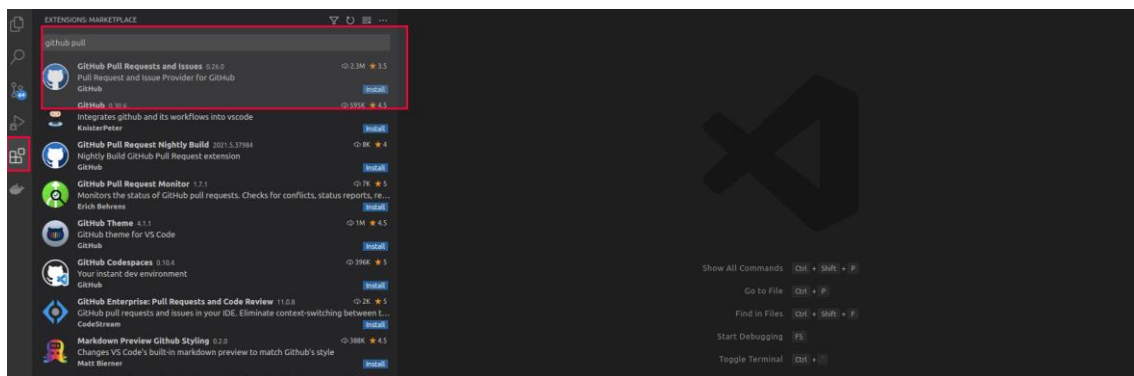
Verificar

Create account

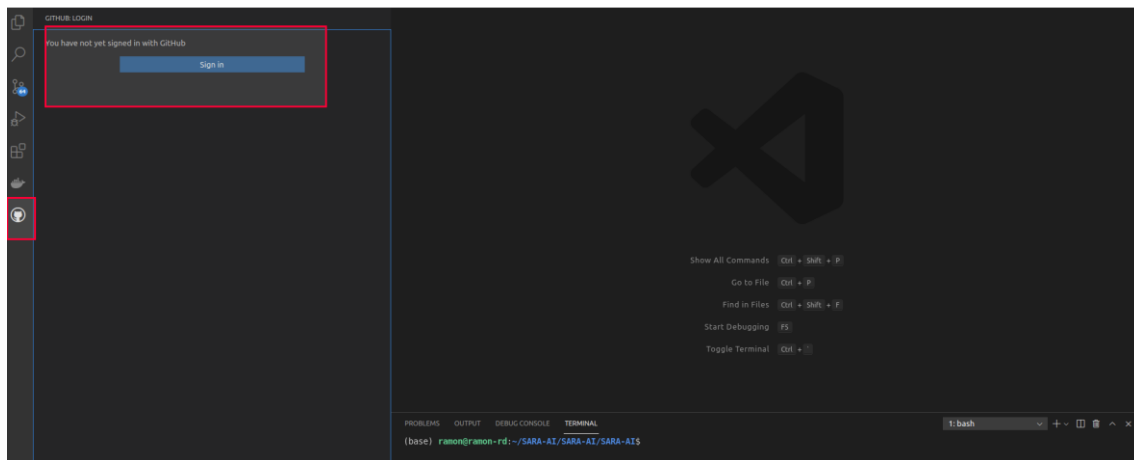
Una vez que hemos creado nuestro usuario y contraseña, podremos acceder a nuestro perfil público de GitHub:



Una vez que hemos creado nuestro usuario en GitHub, procedemos a instalar la extensión “GitHub Pull Requests and Issues” de Visual Studio Code. Para ello, nos dirigimos al apartado extensiones en Visual Studio Code, buscamos e instalamos el paquete mencionado.



Una vez instalado, podremos ver un nuevo icono (de GitHub) en el menú de la izquierda. Nos dirigimos a la extensión de GitHub e ingresamos nuestro usuario y contraseña de GitHub.

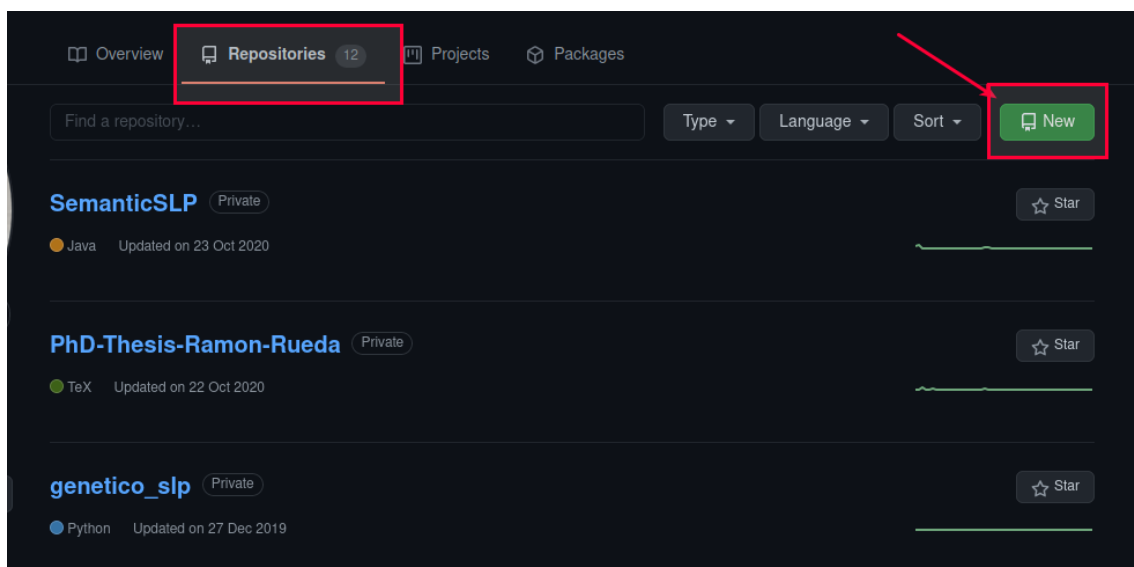


Al hacerlo, aparecerán una serie de ventanas e instrucciones para darle los permisos necesarios a Visual Studio Code para interactuar con GitHub. Siga las instrucciones hasta que finalice el proceso.

2.1 Sincronización proyecto GitHub en Visual Studio Code

Ahora, podemos buscar y clonar un repositorio desde GitHub usando el comando Git. Para llevar a cabo un caso práctico, crearé un repositorio en mi GitHub. Clonaremos dicho repositorio e interactuaremos con él a través de GitHub y Visual Studio Code.

Para crear un nuevo proyecto, nos dirigimos al apartado de repositorios y pulsamos el botón "New":



A continuación le asignamos un nombre a nuestro repositorio. Elegimos si deseamos que el repositorio sea público o privado, y por último elegimos crear el fichero “README” y una licencia para nuestro proyecto. Una vez seguidos estos pasos, pulsamos el botón de crear repositorio.

The screenshot shows the GitHub 'Create a new repository' page. Red annotations highlight specific parts: a red box around the 'Repository name' field containing 'BPP' with a green checkmark; a red arrow pointing to the 'Public' radio button; a red arrow pointing to the 'Add a README file' checkbox; and another red arrow pointing to the 'Choose a license' section, which includes a red box around the 'License: GNU General Public ...' dropdown menu. The 'Create repository' button is at the bottom.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Repository template
Start your repository with a template repository's contents.

No template ▾

Owner * / **Repository name ***

ramon-rd / BPP ✓

Great repository names are short and memorable. Need inspiration? How about [scaling-goggles?](#)

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

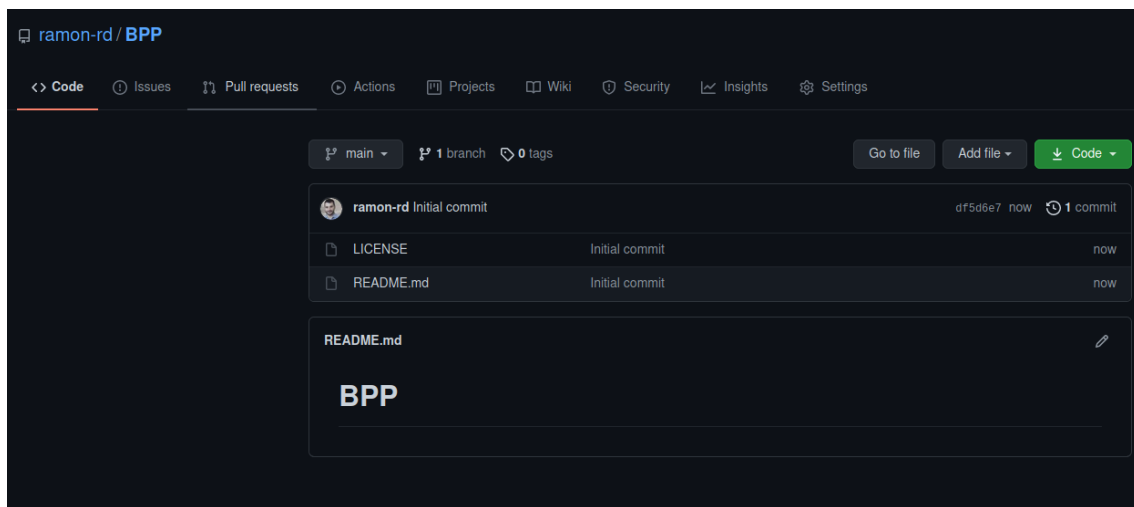
☒ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

License: GNU General Public ... ▾

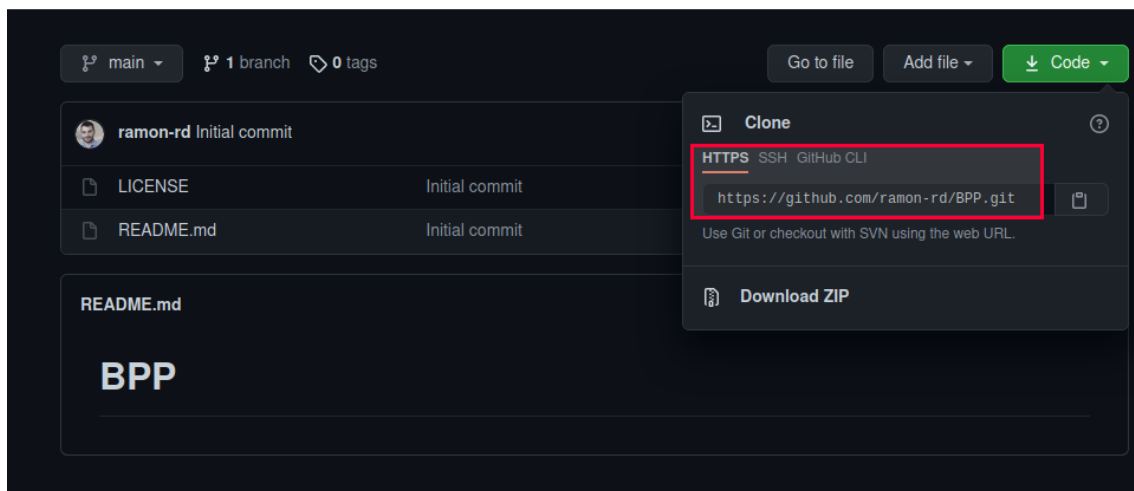
This will set `main` as the default branch. Change the default name in your [settings](#).

Create repository

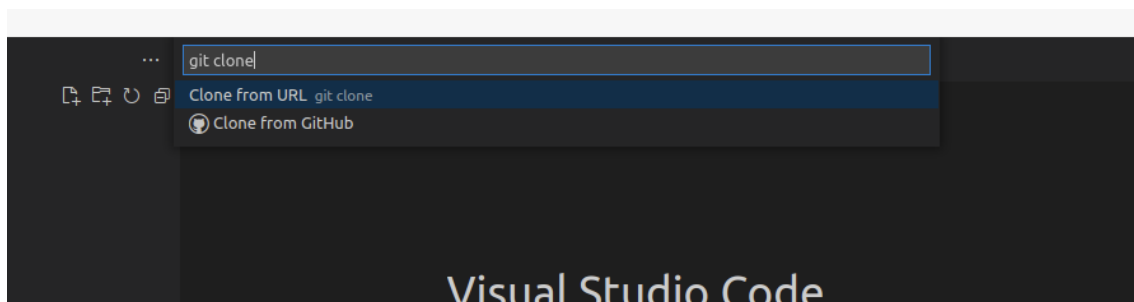
Y listo, ¡habremos creado nuestro primer repositorio!:



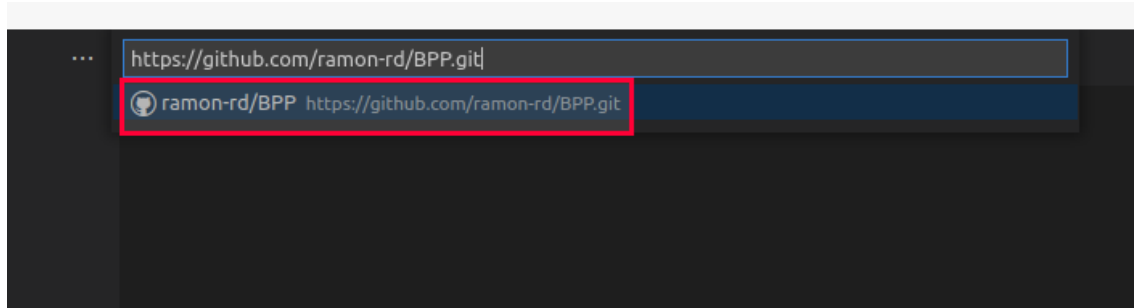
Ahora clonaremos el repositorio en nuestra máquina, haciendo uso de Visual Studio Code. Para ello, nos dirigimos en primer lugar al repositorio creado en GitHub y copiamos el enlace necesario para clonar el proyecto:



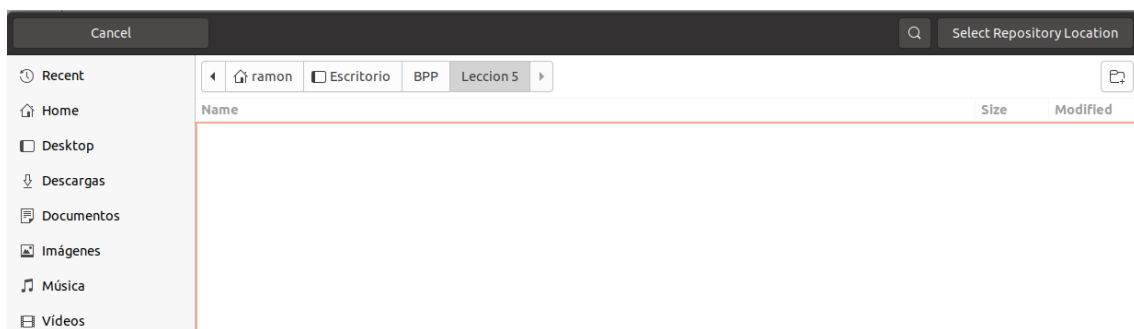
Ahora, con el enlace anterior copiado, nos dirigimos a Visual Studio Code, pulsamos la combinación de teclas Ctrl+Shift+P, y en la paleta de comandos escribimos git clone:



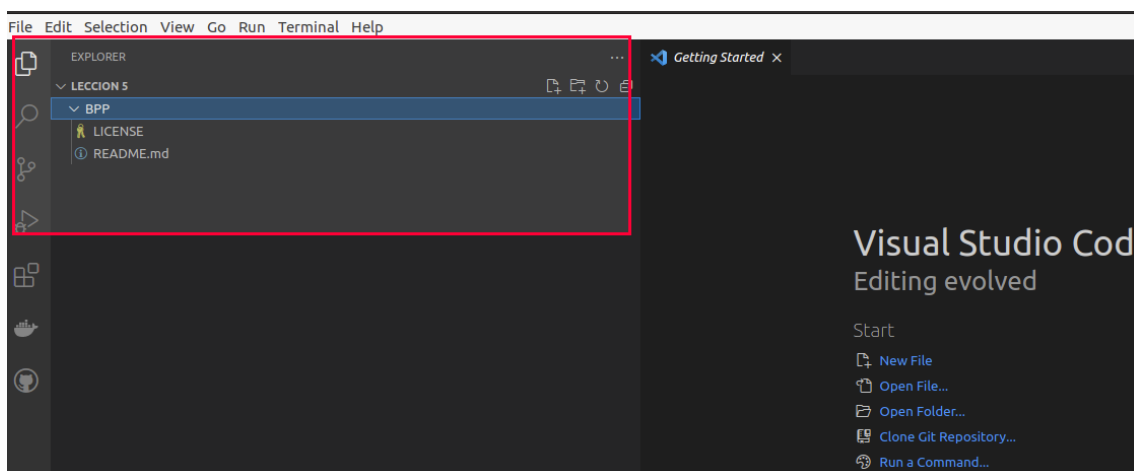
Copiamos el enlace del repositorio:



Y seleccionamos el directorio donde queremos almacenar el proyecto:



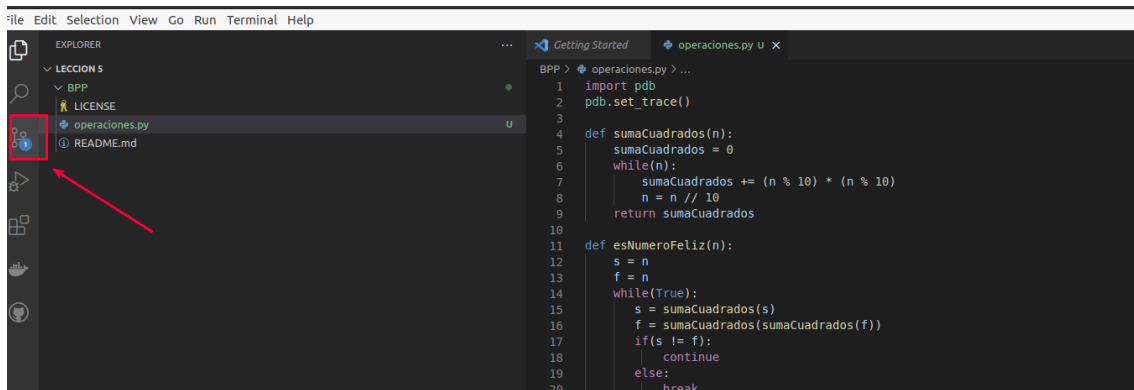
Una vez realizado este proceso, podremos ver en nuestro menú de la izquierda de Visual Studio Code, cómo aparece el proyecto clonado del repositorio junto con los ficheros que se crearon inicialmente: README.md y License.



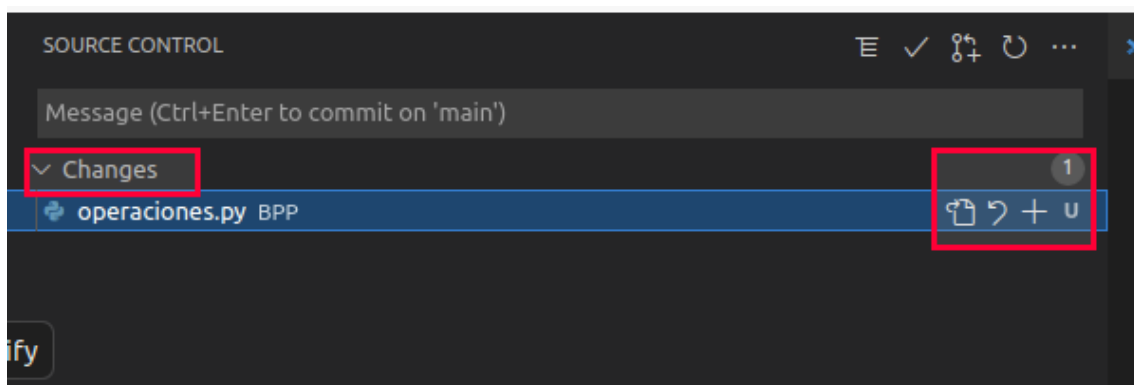
En este punto, podemos trabajar sobre el proyecto que hemos descargado: creando nuevos ficheros de programación e incluyendo contenido sobre estos. Cada vez que hagamos uno o varios cambios, podemos subirlos y registrar sus cambios en GitHub.

2.2 Creación de ficheros y subida a GitHub (commit y push)

En este caso práctico estamos interesados en crear un fichero “.py” que contenga los algoritmos desarrollados en la lección número 4 de esta asignatura. Para ello, creamos un nuevo fichero e incluimos el código:



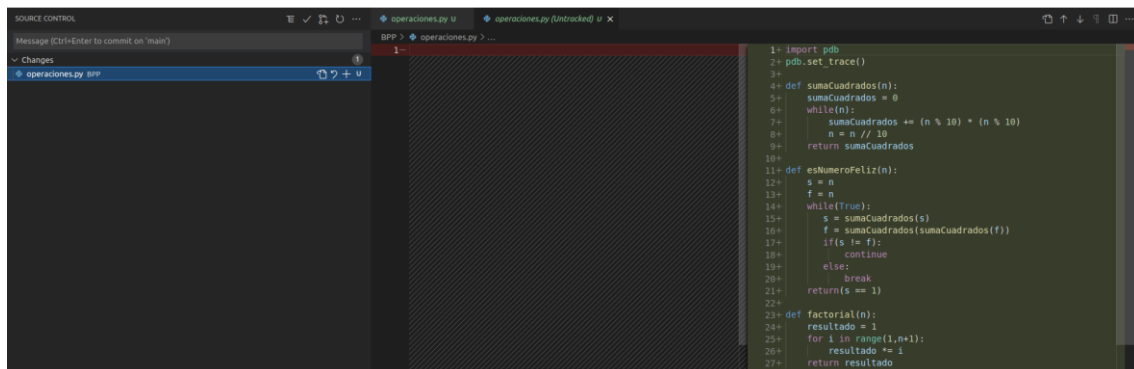
En este punto es importante destacar que cada vez que incluyamos nuevo contenido al proyecto descargado de GitHub, aparecerá una burbuja en el menú de la izquierda indicando que se han detectado nuevos cambios en el proyecto. Para poder interactuar con el repositorio de GitHub, nos dirigimos al menú de la izquierda (señalado con una flecha roja), para ver los cambios que se han realizado:



Dos aspectos importantes a destacar de la captura anterior:

- Tras dirigirnos al menú mencionado anteriormente, podemos ver que se ha detectado un cambio en el fichero operaciones.py.
- A la derecha del nombre del fichero aparece la letra U, referente a update. Esto quiere decir que hay cambios en el código con respecto a lo que hay almacenado en el repositorio de GitHub y que por tanto, debemos actualizar su contenido.

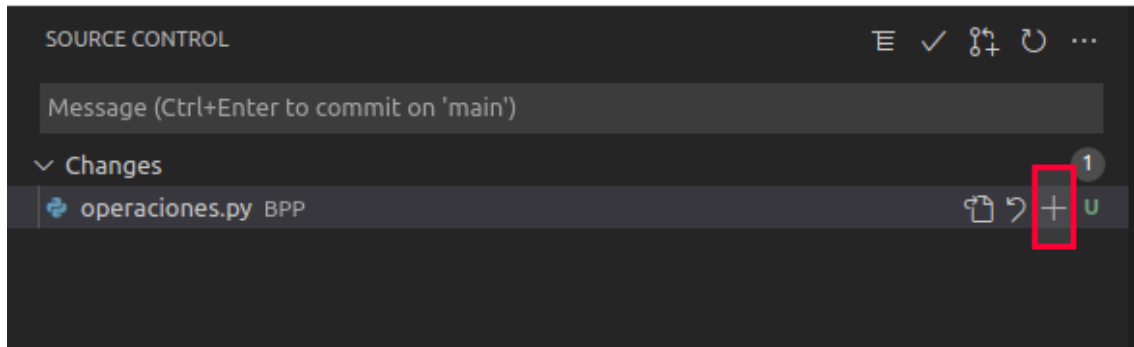
Si pulsamos sobre dicho fichero, comprobaremos que se nos despliega a la derecha dos editores de texto. El de la izquierda muestra el contenido del fichero **ANTES** de hacer los cambios, y a la derecha el estado actual del fichero **DESPUÉS** de haber realizado las modificaciones oportunas.



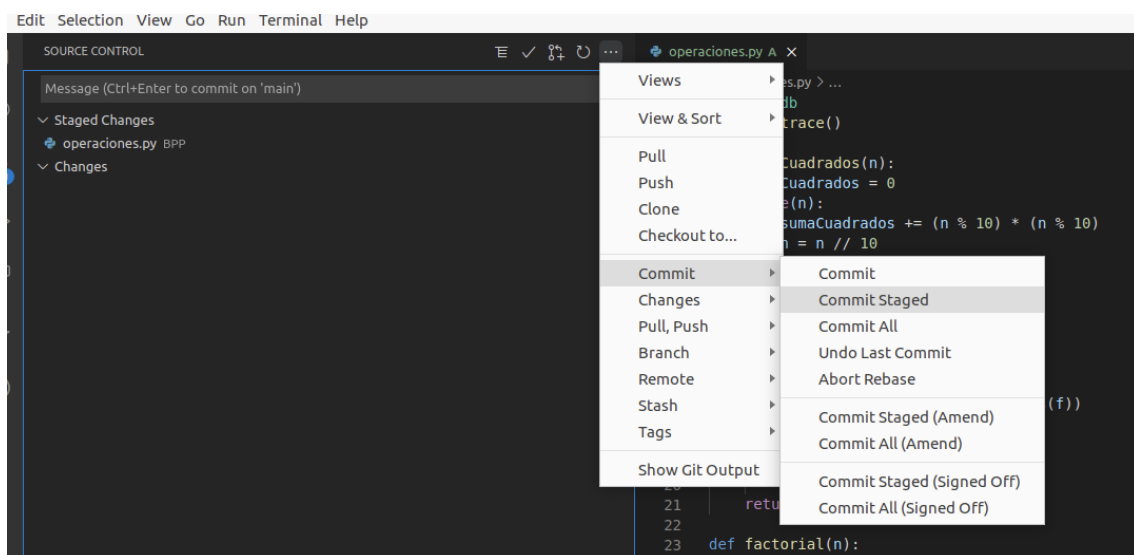
Puesto que el fichero no existía y lo acabamos de crear, Visual Studio Code nos indica (en el editor de la izquierda) que el fichero no contenía nada, y a la derecha todo el contenido creado en verde.

Para realizar una subida de este código a GitHub hemos de realizar los siguientes pasos:

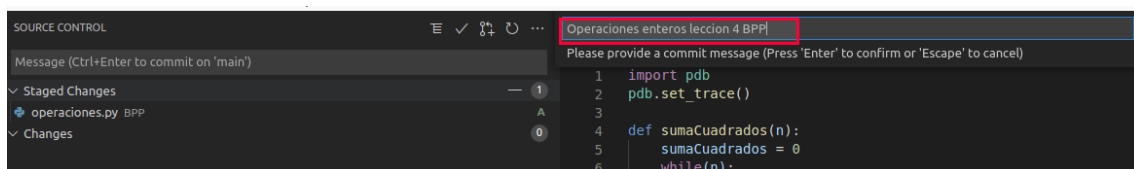
Paso 1: Pulsar el botón “stage changes”



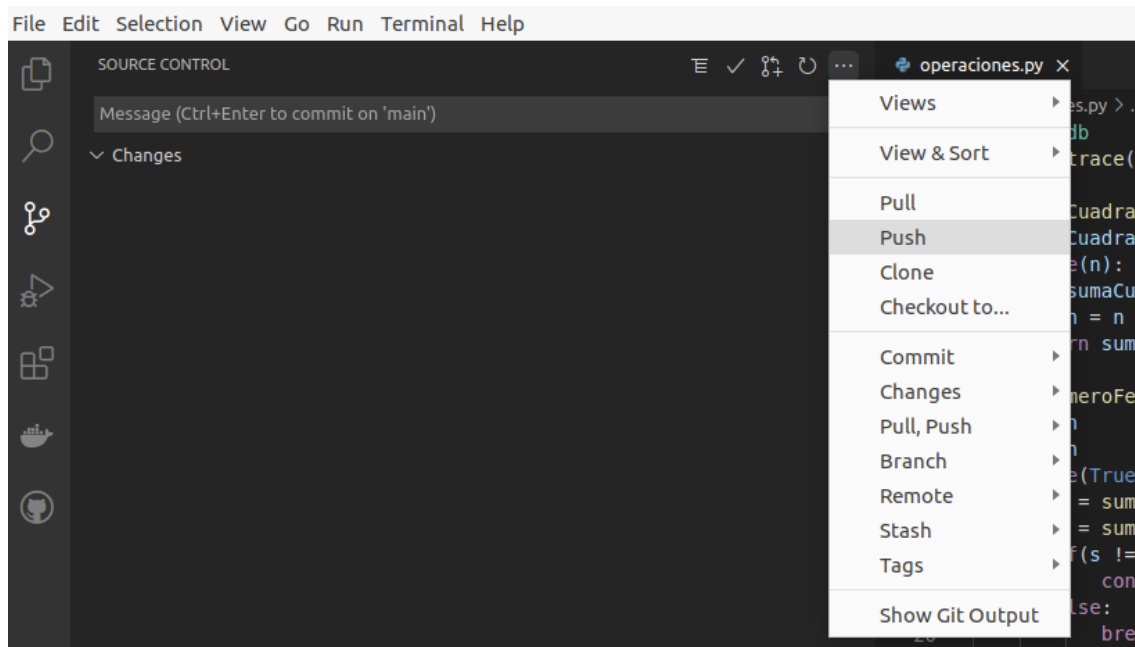
Paso 2: Seleccionamos la opción de “Commit Stages” para describir qué es lo que vamos a subir.



Paso 3: Escribimos un mensaje que describa qué cambios hemos hecho sobre el fichero.

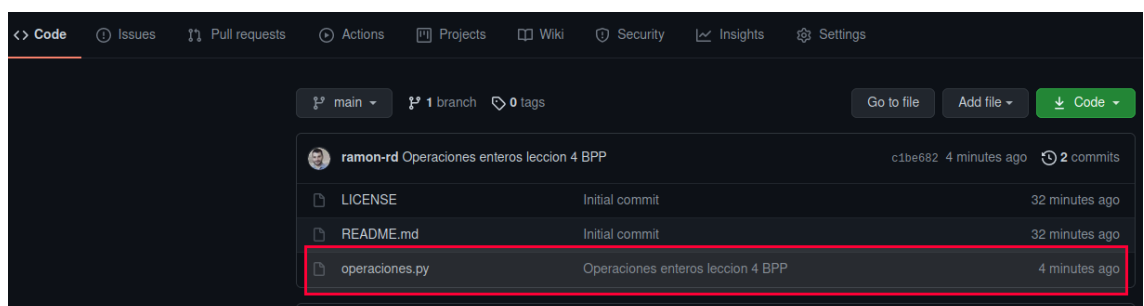


Paso 4: Realizamos la subida pulsando el botón Push.



Tras realizarlo comprobaremos dos cosas:

1. En el menú de la izquierda donde veíamos los cambios que se habían realizado sobre el fichero, no aparece nada.
2. Se ha actualizado el contenido del repositorio de GitHub con todos los cambios que hemos hecho:



Como podréis imaginar en este punto, el código se encuentra replicado en GitHub:

The screenshot shows a GitHub file viewer for the file 'operaciones.py' in the repository 'BPP / operaciones.py'. The file is 45 lines long, 39 sloc, and 868 Bytes. The code is as follows:

```

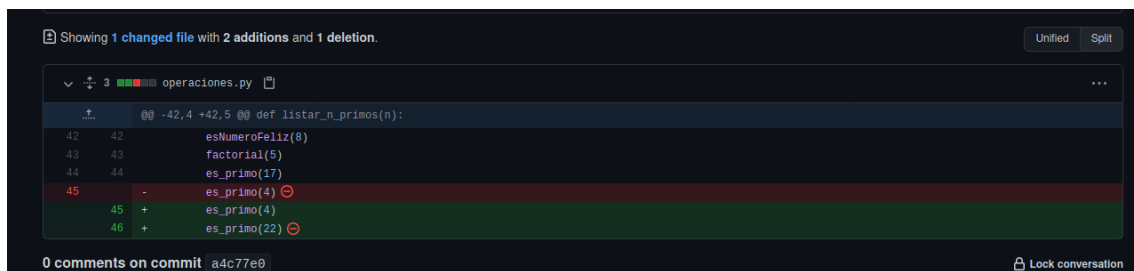
1 import pdb
2 pdb.set_trace()
3
4 def sumaCuadrados(n):
5     sumaCuadrados = 0
6     while(n):
7         sumaCuadrados += (n % 10) * (n % 10)
8         n = n // 10
9     return sumaCuadrados
10
11 def esNumeroFeliz(n):
12     s = n
13     f = n
14     while(True):
15         s = sumaCuadrados(s)
16         f = sumaCuadrados(sumaCuadrados(f))
17         if(s != f):
18             continue
19         else:
20             break
21     return(s == f)

```

Por último, haré unos cambios adicionales al fichero que hemos creado y lo subiré a GitHub. Tras hacerlo (siguiendo el mismo proceso mostrado anteriormente) podremos visualizar las diferentes versiones que hemos ido teniendo de un mismo fichero desde su creación:

The top screenshot shows the same file viewer as before, but with the 'History' button highlighted in a red box. The bottom screenshot shows the 'History' view for the file 'operaciones.py'. It displays two commits by 'ramon-rd' on May 28, 2021:

- Commit 'a4c77e0' titled 'Comprobar si 22 es primo', committed 4 minutes ago.
- Commit 'c1be682' titled 'Operaciones enteros leccion 4 BPP', committed 18 minutes ago.



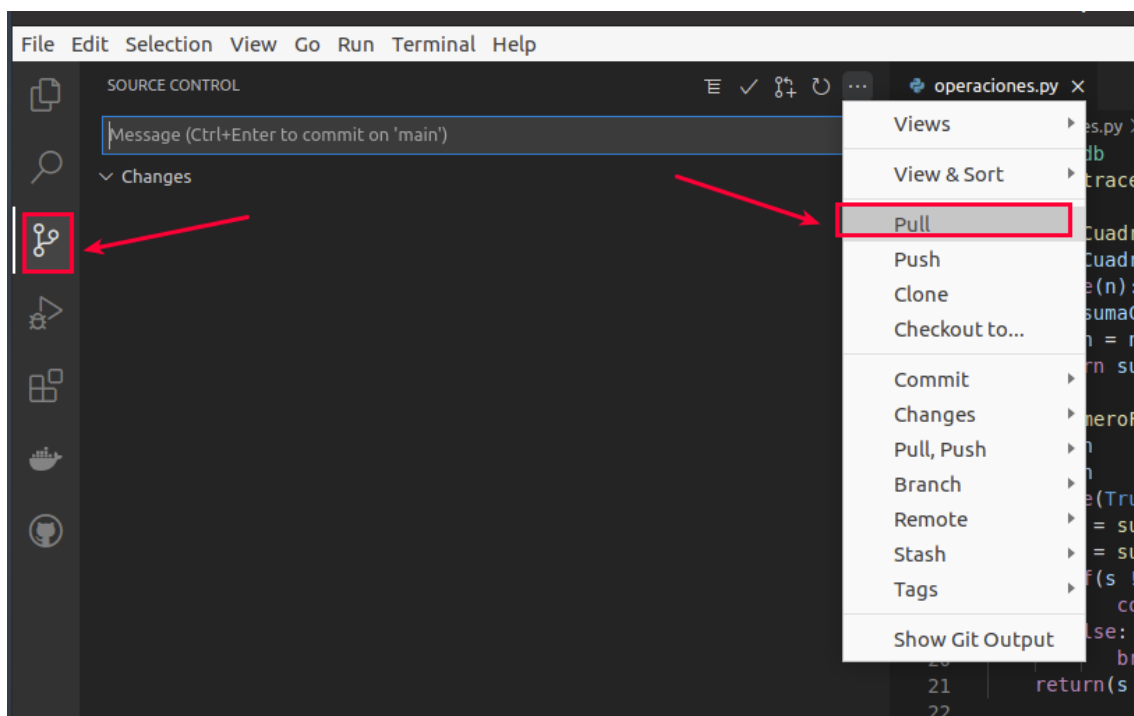
```
Showing 1 changed file with 2 additions and 1 deletion.
Unified Split

operaciones.py
@@ -42,4 +42,5 @@ def listar_n_primos(n):
42 42     esNumeroFeliz(8)
43 43     factorial(5)
44 44     es_primo(17)
45 -     es_primo(4)
45 +     es_primo(4)
46 +     es_primo(22)
```

Nótese que el último cambio realizado se muestra en la historia de la última versión del fichero. En rojo se muestran las líneas que he eliminado y en verde el nuevo contenido creado.

2.3 Descarga y actualización del proyecto (Pull)

Si estamos trabajando de forma colaborativa, debemos descargar todos los cambios que se hayan producido en el proyecto e incorporarlos en nuestro directorio local. Para ello usaremos la orden pull. Este comando comprueba si la versión alojada en local coincide con la que hay en el repositorio de GitHub, en caso de que no coincidan, se descarga y actualiza el contenido en local:



3. CONCLUSIONES GENERALES

Conocer y comprender con profundidad el funcionamiento de GitHub requiere una inversión importante de tiempo. Sin embargo, dedicar unos primeros esfuerzos a conocer en detalle GitHub (o al menos sus herramientas más utilizadas) marcarán una gran diferencia a la hora de desarrollar un proyecto software y trabajar en equipo con otros usuarios. El objetivo de esta lección ha sido el de describir los Sistemas de Control de Versiones y mostrar una visión general sobre cómo utilizarlos, haciendo hincapié en las principales operaciones que se utilizan en el día a día.

No obstante, si le resulta de interés y quiere profundizar más sobre su uso, se recomienda visitar la web oficial de GitHub, en la cual podemos encontrar una gran cantidad de documentación para comprender su uso con el máximo nivel de detalle:

<https://docs.github.com/en/github/getting-started-with-github/quickstart>

4. PUNTOS CLAVE

En esta lección hemos aprendido:

- Qué son y para qué sirven los sistemas de control de versiones.
- Cuáles son las principales características de los sistemas de control de versiones.
- Cómo integrar GitHub en nuestro IDE Visual Studio Code.
- Cómo interactuar GitHub desde Visual Studio Code.

